COS 598 Final Project Report: XGBoost: An Empirical Comparison of Traditional & Industry Machine Learning Methods

Christopher Ewanik
Saher Fatima
Dorothy Harris

May 5, 2023

Abstract

XGBoost is considered one of the leading machine learning algorithms for regression and classification and is used by data scientists worldwide. Due to its high accreditation, this study tests XGBoost's performance against its common academia counterparts by both performance and training time. This study was conducted by comparing XGBoost's classification model to Support Vector Classification (SVM) and Naive Bayes, and XGBoost's regression model to Kernel Ridge Regression (KRR) and Elastic Net. The hyperparameters are tuned with a Tree-Structured Parzen Estimator (TPE) using a library called Optuna. For classification, the default XGBoost had the best testing accuracy at 0.87. For Regression, the Optuna tuned KRR had the best testing MSE at 0.13. The TPE was highly effective for tuning hyperparameters, particularly with the Elastic Net Regression. The SVM took over an hour to optimize hyperparameters and, despite more parameters, XGBoost optimized relatively quickly.

1 Introduction

This project aimed to compare the speed and performance of XGBoost with traditional academic machine learning methods. The paper, "XGBoost: A Scalable Tree Boosting System" claimed to have developed an algorithm that runs over ten times faster than existing popular solutions. The problem with the paper is that it uses massive datasets and its performance is only compared with other major tree-boosting systems. We intended to test these findings and see if XGBoost was truly the optimal method when compared to traditional machine learning algorithms.

The motivation behind this paper was to conduct a study comparing XGBoost to traditional machine learning algorithms. While efficiency and accuracy on large datasets are great, this study aimed to see if competitive benefits existed on small to medium datasets when compared to traditional algorithms. This paper specifically focuses on XGBoost's performance and speedup capabilities for classification and regression problems, as these were the methods discussed in class. The goal was to see how XGBoost performs in terms of training speed and model performance compared to classic machine learning methods. In order to compare how to quantify XGBoost's classification performance, this paper tested it against a SVM model as well as a Naive Bayes model. To measure XGBoost's regression performance, it was tested against both Elastic Net and KRR models. In all the models, the hyperparameters were tuned by a Tree-Structured Parzen Estimator (TPE) using a library called Optuna.

Since this project explores both classification and regression, two different datasets from UC Irvine were used. For the classification models, the Adult dataset was chosen. This dataset was chosen due to its large size and number of categorical attributes that can be one hot encoded. The target data from this dataset was a binary variable indicating whether income is above or below $50,000. The regression models used the Auto MPG dataset. This dataset is a multivariate regression with features describing parts of automobiles and is commonly used for regression models to predict the continuous target variable of miles per gallon (mpg). The dataset was chosen partially due to its smaller size which allowed for a nice comparison to the Adult dataset.

2 Related Work

Due to the popularity of XGBoost, many papers have been written about it. Dwidarma et al. [8] compared XGBoost to logistic regression for use in predicting potential debtors. They found that XGBoost had better results than logistic regression when compared using accuracy, sensitivity, precision, and specificity. Munkhdalai et a. [4] compared XGBoost to logistic regression, MARS, SVM, Random Forest, and ANN. They tested on AUC, h-measure, true positive rate, false positive rate, and accuracy. The group found that XGBoost resulted in the best accuracy and false positive rate. Santhanam et al. [7] tested XGBoost's accuracy and speed with a prediction of multiple datasets. The individuals also tested the regression capabilities using MSE and classification capabilities using accuracy. Binson et al. [9] used XGBoost and SVM to predict pulmonary diseases. They found an accuracy of 91.74%, 89.84%, and 70.66% for each of their predictions. Like many of these papers, we used the accuracy metric for classification, but we also tested XGBoost's regression capabilities using MSE and speed of training and testing. We also tested against many models while many of these papers just tested against one.

3 Methodologies

This section describes the methodologies explored for this project as well as the rationale behind them. SVM classification, KRR regression, and Naive Bayes classification were also used in this project, but will not be explained here since they were discussed in class.

3.1 Elastic Net Regression

Elastic net regression aims to select the predictor variables most important for predicting the target variable while using regularization to avoid overfitting the model. The Elastic Net penalty is a weighted combination of the L1 and L2 penalties, where a hyperparameter called λ controls the weight. When λ is 0, the elastic net penalty reduces to Ridge regression, and when λ is 1, it reduces to Lasso regression.

Elastic net regression estimates the coefficients $\beta_0, \beta_1, \ldots, \beta_p$ by minimizing the following objective function.

$$\frac{1}{2n} \left| y - Xw \right|_2^2 + \alpha\lambda \left| w \right|_1 + \frac{\alpha(1-\lambda)}{2} \left| w \right|_2^2 \tag{1}$$

Where y is a vector representing the target variable, X is an $n \times p$ matrix of predictor variables, w is a $p \times 1$ vector of coefficients, n is the number of samples, α is a constant that multiplies the penalty terms, and λ is the Elastic Net mixing parameter that, if set to zero, the equation uses L2 regularization and if 1 uses L1 regularization [6].

3.2 XGBoost

XGBoost is a decision-tree-based ensemble algorithm that uses gradient boosting and hardware optimizations to produce quick, accurate results [3]. The algorithm is effectively used for regression, classification, ranking, and time series tasks. This section aims to provide the reader with a rough understanding of how the algorithm works. Pieces of the paper are summarized briefly below, but the reader is encouraged to read the full publication for a thorough understanding.

3.2.1 Regularized Learning Objective

Suppose we have data in the following form with n examples and m features $D = (x_i; y_i)$ $(|D| = n; x_i \in R^m; y_i \in R)$. In general, the goal is to minimize the following regularized objective.

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{2}$$

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda |w|_2^2 \tag{3}$$

Where l is a differentiable convex loss function measuring the difference between the predicted and the actual, Ω acts as the regularization term penalizing the complexity of the tree, T is the number of leaves in the tree, f_k represents each unique weak learning tree with leaf weights w, and γ penalizes the number of leaves. When this regularization parameter is set to zero, the objective simply becomes traditional gradient tree boosting.

3.2.2 Gradient Tree Boosting

The reader should be aware of two more equations, the first being the equation for the optimal value. The optimal value is used as a score function to measure the quality of a tree structure q. XGBoost uses this equation to identify attributes that produce the purest splits. Note that g_i and h_i are the first and second order gradient statistics on the loss function.

$$L(t)(q) = \frac{1}{2} \sum_{j=1}^{T} \left(\frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} \right) + \gamma T \tag{4}$$

The following equation is used to calculate the loss reduction after a split. Assume that I_L and I_R are the instance sets of left and right nodes and that $I = I_L \cup I_R$.

$$L_{\text{split}} = \frac{1}{2} \left(\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} \right) - \gamma \tag{5}$$

3.2.3 Exact Greedy Algorithm and Approximate Greedy Algorithm

Greedy algorithms are used to help XGBoost determine when to split. The Exact Greedy Algorithm for Split Finding, uses Eq. (5) and enumerates over all possible splits on all the features. This allows XGBoost to determine all optimal splits and on what features.

3.2.4 Sparsity Aware Split Finding

Datasets often have a sparse feature space which is difficult for some machine learning algorithms to handle. Sparse data is often the result of missing data, zero entries in statistics, or feature engineering techniques, such as one-hot encoding. To deal with this, the authors added a default direction in each tree node which was learnt from the data and proposed their own sparsity-aware split finding algorithm.

3.2.5 Parallel Learning, Cache Aware Access and Blocks for Out of Core Computation

XGBoost implements a variety of hardware optimizations that enhance its speed. These methods include block structure for parallelization, a cache-aware prefetching algorithm, block compression and sharding. These methods are crucial for the efficient use of the exact greedy algorithm.

3.2.6 Differences between Regression and Classification Tree

Overall the regression and classification models are substantially similar. The only true difference between the two methods is their loss function. Regression problems typically use a loss like MSE or RMSE, while classification problems use logistic loss for binary classification or the softmax function for multiclass classification.

3.3 Optuna

In the paper, Optuna: A Next-generation Hyperparameter Optimization Framework [1], Akiba et al. detail the advanced library, Optuna. Optuna is an optimization library that enables users to use state-of-the-art algorithms to optimize and drastically reduce the time it takes to tune hyperparameters.

3.3.1 Tree-Structured Parzen Estimator

Sampling is the act of creating new hyperparameter combinations to evaluate during optimization. This study used the independent sampling technique, Tree-Structured Parzen Estimator (TPE). Originally proposed in the 2011 paper by Bergstra, Bardenet, Bengio (the lead designer of Theano and recipient of the 2018 Turing Award) and Kégl [2], TPE models $p(x|y)$ by replacing the distributions of the prior configuration with non-parametric densities. Generally speaking, TPE substitutes an equally-weighted mixture of the prior with Gaussians centered at each of the features.

3.3.2 Median Pruning

Pruning terminates trials early during training and is crucial in maintaining the cost-effectiveness of optimization. Overall, pruning watches objective function scores and terminates attempts that do not meet specified conditions [5]. This study was elected to use median pruning. Median pruning terminates if the trial's best intermediate result is worse than the median of intermediate results of previous trials at the same step.

3.4 Data Preprocessing

XGBoost claims to have superior performance on larger sparse datasets created from feature engineering. Therefore, this paper preprocesses the data in a way that could potentially show performance differences between different algorithms and XGBoost. Overall, all rows with NAs are first removed, as well as any redundant columns. Next, a standard scalar is applied to the continuous variables. In the auto data set, the car name is split into "make" and "model". The most significant change is using one-hot encoding on all categorical columns. This results in addition of more dimensions that contain lots of sparse data. The adult dataset goes from 32561 rows and 15 columns to 30162 rows and 103 columns, while the auto dataset changes from 390 rows with 10 columns to 390 rows with

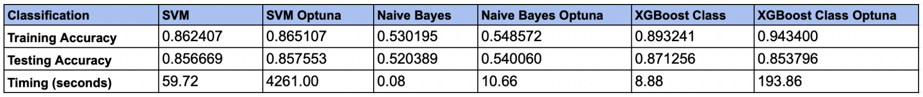
26 columns. These choices were made with the hope of testing XGBoost's ability with smaller datasets and challenging the traditional algorithm's ability to deal with sparsity. Data is split into training and testing sets using 30% of the data for testing.

4 Results and Discussion

4.1 Model Setup

All the models were trained using Google Colab standard CPUs. To get an accurate comparison of all the models, each model was tested using the same metric. For classification, models were evaluated using accuracy and for regressions the models were evaluated using mean squared error (MSE). All models were timed to see how long it took to optimize hyperparameters. The study used 100 Optuna trials on each model to find the optimal hyperparameters, which also provided a standard unit to assess time to optimize.

4.2 Classification Results and Discussion



Classification	SVM	SVM Optuna	Naive Bayes	Naive Bayes Optuna	XGBoost Class	XGBoost Class Optuna
Training Accuracy	0.862407	0.865107	0.530195	0.548572	0.893241	0.943400
Testing Accuracy	0.856669	0.857553	0.520389	0.540060	0.871256	0.853796
Timing (seconds)	59.72	4261.00	0.08	10.66	8.88	193.86

Figure 1: Classification models training and testing accuracy and timing results.

Figure 1 and 2, show the test results for all of the classification models before and after hyperparameter tuning using Optuna. The greatest accuracy was achieved with the default XGBoost model with an accuracy of around 87%. The tuned XGBoost model achieved a 94% training accuracy but ultimately overfit as there was a fairly significant drop off in test accuracy. The Naive Bayes model struggled greatly with this dataset. The general theory as to why is that, the mix of one hot encoded features and continuous features was not a good match for Gaussian Naive Bayes, which is generally better suited for only continuous data. There could also be some violations of the independence assumption in this dataset as each row roughly represents a demographic of the population. The SVM performed closely to the XGBoost model with a testing accuracy of 85%. It was interesting to note that tuning the hyperparameters had no significant influence on this model's performance. The drawback of the SVM was the massive training time. Tuning the hyperparameters of the SVM took over an hour. Meanwhile, the XGBoost model took just over three minutes. This difference appears to be evidence of the sparsity-aware algorithm and the hardware optimizations that XGBoost implements. Naive Bayes fits exceptionally fast, but considering the poor performance, it is not likely a viable option.

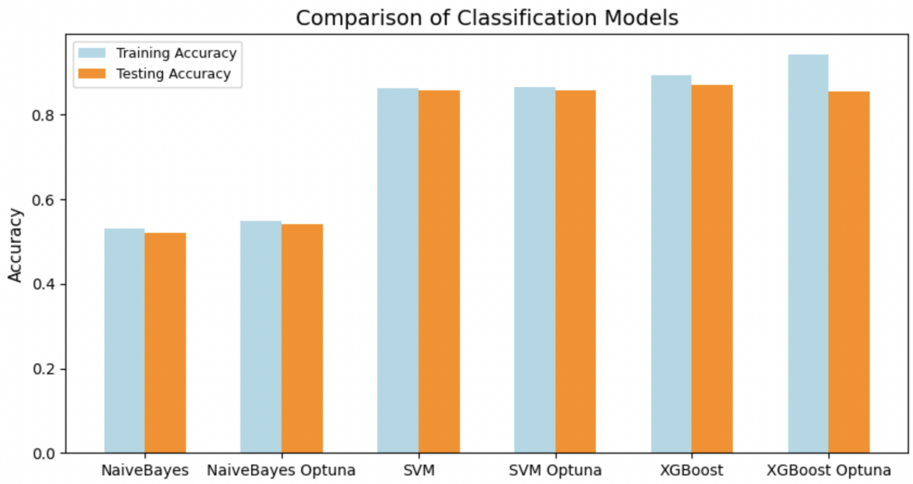
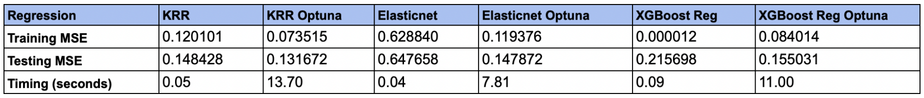


Figure 2: Comparison of Classification Models.

4.3 Regression Results and Discussion

In figures 3 and 4, the tuned KRR performed best among all models with a low test MSE of 0.13, although the tuned Elastic Net and XGBoost model did not lag far behind. It was interesting to see the strength of the default XGBoost regression although it ultimately lead to overfitting. The Elastic Net regression greatly benefited from optimizing the L1 and L2 regularization penalties. This was the most drastic improvement seen with the use of Optuna. All of the models were optimized and trained in a timely manner, which was expected given the small number of rows in the dataset.



Regression	KRR	KRR Optuna	Elasticnet	Elasticnet Optuna	XGBoost Reg	XGBoost Reg Optuna
Training MSE	0.120101	0.073515	0.628840	0.119376	0.000012	0.084014
Testing MSE	0.148428	0.131672	0.647658	0.147872	0.215698	0.155031
Timing (seconds)	0.05	13.70	0.04	7.81	0.09	11.00

Figure 3: Regression models training and testing MSE and timing results.

5 Conclusion

This study provided reliable answers to the questions not resolved in the original paper. When using Optuna and a Tree-Structured Parzen Estimator algorithm to tune the hyperparameters, there was no significant evidence that XGBoost provided meaningful advantages over traditional algorithms on small to medium sparse datasets. Support Vector Machine took a dramatic amount of time to tune

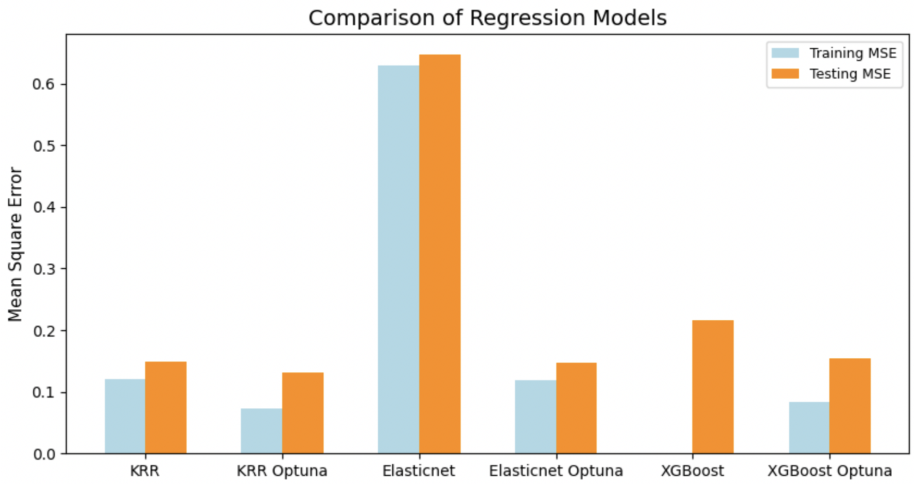


Figure 4: Comparison of Regression Models; Note that the XGBoost train MSE is visible but extremely small.

it's hyperparamters and Elastic Net regression benefited greatly from optimizing the L1 and L2 tradeoff. Optuna and TPE proved effective in improving model performance, boosting all but one model in testing accuracy. Optuna also proved highly efficient scanning through huge ranges of hyperparameters that would have taken dramatically longer with traditional grid searches.

6 Description of Individual Effort

For this project, each one of us worked on a specific classification and regression algorithm. Saher worked on setting up the KRR regression model, the Naive Bayes classification model and constructed the graphs/tables. Dorothy worked on setting up the XGBoost classification model and the SVM classification model. Chris was in charge of the XGBoost regression model, the Elastic Net regression model and worked on preprocessing the data. Each member of the group implemented Optuna into their respective models.

References

[1] Takuya Akiba, Shotaro Sano, Takeru Yanase, Tomohiro Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *Journal of Machine Learning Research*, 20(124):1–8, 2019.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[4] Oyun-Erdene Namsrai Jong Yun Lee Lkhagvadorj Munkhdalai, Tsendsuren Munkhdalai and Keun Ho Ryu. An empirical comparison of machine-learning methods on bank client credit assessments. In *MDPI*, 2019.

[5] Optuna. optuna.pruners.medianpruner – optuna documentation. `https://optuna.readthedocs.io/en/stable/reference/generated/optuna.pruners.MedianPruner.html`, n.d. Retrieved May 5, 2023.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. sklearn.linear_model.elasticnet. `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html`, 2021. Accessed: May 5, 2023.

[7] Sunil Raman Ramraj Santhanam, Nishant Uzir and Shatadeep Banerjee. Experimenting xgboost algorithm for prediction and classification of different datasets. In *International Journal of Control Theory and Applications*, 2017.

[8] Syarifah D. Permai Revina Dwidarma and Jeklin Harefa. Comparison of logistic regression and xgboost for predicting potential debtors. In *AiDAS*, 2021.

[9] Youhan Sunny V. A. Binson, M. Subramoniam and Luke Mathew. Prediction of pulmonary diseases with electronic nose using svm and xgboost. In *IEEE Sensors*, 2021.